Digital logic Design

unit-I Number systems, codes, Boolean Algebra

i) Representation of signed Numbers -

ii) Binary arithmatic in Computers

CODES :-

i) weighted and Non weighted codes.

ii) Sequential codes

iii) self Complementing codes

iv) cyclic codes

(v) The 8421 BCD codes, BCD addition

(iv) Excess-3 code

(vii) The Gray codes - Binary to Gray & Gray to Binary converter

(viii) Error detecting codes: Parity, check sums, Block parity, Five-bit Codes, The Biquinary code. The Ring Counter code

(ix) Error Correcting Code: 7 bit Hamming code:

(x) Alphanumeric Codes: The ASCII Code, The EBCDIC Code.

Boolean Algebra

i) Logic operations;

ii) Axioms and Laws of Boolean Algebra: Complementation Laws, AND Laws, OR Laws Commutation laws, Associative Laws, Distributive Laws, Redundant Literal Rule, Idempotence Laws, Absorption Laws, Transposition Theorem, Demorgan's Theorem; Duality;

iii) Reducing Boolean Expressions;

(iv) Functionally Complete sets of operation

(v) Boolean Functions and Their Representation.

Digital logic Design

unit-I Number systems, codes, Boolean Algebra

i) Representation of signed Numbers -

ii) Binary arithmatic in Computers

CODES :-

i) weighted and Non weighted codes.

ii) Sequential codes

iii) self Complementing codes

iv) cyclic codes

(v) The 8421 BCD codes, BCD addition

(iv) Excess-3 code

(vii) The Gray codes - Binary to Gray & Gray to Binary converter

(viii) Error detecting codes: Parity, check sums, Block parity, Five-bit Codes, The Biquinary code. The Ring Counter code

(ix) Error Correcting Code: 7 bit Hamming code:

(x) Alphanumeric Codes: The ASCII Code, The EBCDIC Code.

Boolean Algebra

i) Logic operations;

ii) Axioms and Laws of Boolean Algebra: Complementation Laws, AND Laws, OR Laws Commutation laws, Associative Laws, Distributive Laws, Redundant Literal Rule, Idempotence Laws, Absorption Laws, Transposition Theorem, Demorgan's Theorem; Duality;

iii) Reducing Boolean Expressions;

(iv) Functionally Complete sets of operation

(v) Boolean Functions and Their Representation.

QUALCOMM

Digital electronics – Digital electronics or digital circuits are electronics that operate on digital signals. Branch of

Digital signal – A signal that is sampled at discrete points and the sampled values are quantized into a set of discrete values.
At a given time it can only take on one of a finite no. of values.

Analog signal – In this signal we get its value at any instant of time.

Discrete signal – we get its value at a particular instant of time (not any time).

Number system – In DEs, the no. system is used for representing the information.

The NS has different bases and the most common of them are Decimal, Binary Octal and Hexadecimal.

The base or Radix of the NS is the total numbers of the digit used in the NS.

for example –

In Decimal NS	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	= 10
Binary	0, 1	= 2
Octal	0, 1, 2, 3, 4, 5, 6, 7	= 8
Hexadecimal NS	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	= 16

1) to communicate 2) to perform tasks
3) to quantify 4) to measure

NS in use – i) Symbolic no. system –
uses Roman Numerals
(I = 1, V = 5, X = 10, L = 50, C = 100, D = 500
M = 1000)

ii) Weighted position system –
Decimal, Binary, Octal, Hexa

The value of a number is: weighted sum of its digits.

For ex:– $2357 = 2\times10^3 + 3\times10^2 + 5\times10^1 + 7\times10^0$

$526.47 = 5\times10^2 + 2\times10^1 + 6\times10^0 + 4\times10^{-1} + 7\times10^{-2}$

10 is called the Base of the DNS

General positional number system –
The value of the number is

$$\boxed{D = \sum_{i=-n}^{p-1} d_i r^i}$$

d_i = digit , r = radix

Binary Number System –
its Base / radix is 2
Therefore it has only two numberals, 0 & 1

For ex:– $(X)_2 = (11110010)_2$
eight digit / bit binary number
binary digits are also known as bits
$(X)_2$ is an 8-bit number

Binary no. to Decimal number –

$(X)_2 = (11110010)_2$

$= (1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 \times 1\times2^1 + 0\times2^0)_{10}$

$(X)_2 = (128+64+32+16+0+0+2+0)_{10}$

$(X)_2 = (242)_{10}$

QUALCOMM

Binary fractional no. to decimal number

$(X)_2 = 101 \cdot 101$

$(X)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

$(X)_2 = (4 + 0 + 1 + \frac{1}{2} + 0 + \frac{1}{8})_{10} = (5 + 0.5 + 0.125)_{10}$

$(X)_2 = (5.625)_{10}$

Features of Binary numbers –

(i) It requires very long strings of 1s & 0s

(ii) Some simplification can be done through grouping

I. 3 bit grouping – octal (radix 8)

The digits will have one of the eigth values from 0, 1, 2, 3, 4, 5, 6, 7

II. 4 bit grouping – By grouping 4 binary digit we form a set Hexadecimal number (Base-16) and the digit will have one of the sixteen values of 0 through 15

Decimal values from 10 to 15 are designated as

A(=10), B(=11), C(=12), D(=13), E(=14), F(=15)

1010, ~~1010~~ 1011, 1100, 1101, 1110, 1111

Conversion of Binary Numbers –

Binary number to Octal Number

Step I Group the binary digits into groups of three

II write each group of three digits to a equivalent decimal number.

$(11011001)_2 = [(011)(011)(001)] = (331)_8$

Binary number to Hexadecimal number

Step I Group the Binary digits into groups of four

II write equivalent decimal no. of each group of four digit.

$(11011001)_2 = [(1101)(1001)]_2 = (D9)_{16}$

QUALCOMM

Changing the Radix of numbers.

$D = \sum_{i=-n}^{p-1} d_i r^i$ The decimal equivalent value of a number in any radix

Octal no. to Decimal Number

I $(331)_8 = (3\times8^2 + 3\times8^1 + 1\times8^0)_{10} = (192+24+1)_{10}$
$= (217)_{10}$

II $(33.56)_8 = (3\times8^1 + 3\times8^0 + 5\times8^{-1} + 6\times8^{-2})_{10}$
$= (24+3+0.625+0.09375)_{10}$
$(33.56)_8 = (27.71875)_{10}$

Hexa-decimal to Decimal number

I $(D9)_{16} = (13\times16^1 + 9\times16^0)_{10} = (208+9)_{10} = (217)_{10}$

II $(E5.A)_{16} = (14\times16^1 + 5\times16^0 + 10\times16^{-1})_{10}$
$= (224+5+0.625)_{10} = (229.625)_{10}$

Conversion of decimal numbers to any numbers with radix r

~~Step I~~ To convert a no. with radix r to a decimal no

Step I Divide the number by r

II Remainder : d_0

III Quotient $Q_1 = ((\cdots((d_{n-1})\cdot r + d_{n-2})r + \cdots)\cdot r + d_1$

III Division of Q_1 by r gives d_1 as the remainder & so on

Represent a no. with radix r as

$D = ((\cdots((d_{n-1})\cdot r + d_{n-2})r + \cdots)\cdot r + d_1)\cdot r + d_0$

Decimal to Binary Conversion

$(156)_{10} = (N)_2$

I

	Quotient	Remainder
$156 \div 2$	78	0
$78 \div 2$	39	0
$39 \div 2$	19	1
$19 \div 2$	9	1
$9 \div 2$	4	1
$4 \div 2$	2	0
$2 \div 2$	1	0
$1 \div 2$	0	1

Upward direction

$(156)_{10} = (10011100)_2$

Decimal to octal number Conversion

$(678)_{10} = (N)_8$

	Quotient	Remainder
$678 \div 8$	84	6
$84 \div 8$	10	4
$10 \div 8$	01	2
$1 \div 8$	0	1

$(678)_{10} = (1246)_8$

Decimal to Hexa-decimal number Conversion

$(678)_{10} = (N)_{16}$

	Quotient	Remainder
$678 \div 16$	42	6
$42 \div 16$	2	A
$2 \div 16$	0	2

$(678)_{10} = (2A6)_{16}$

QUALCOMM

Negative Numbers

(2) I. Sign-magnitude representation:- an additional "+" sign before a number indicates that it is a positive number
"−" sign before a number indicates that it is a negative number

→ but called sign bit is placed in front of the number. → Representation of negative no. is not very convenient on computers.

Sign magnitude NS is possible, it is impractical.
→ Therefore "+" sign is replaced by 0 & "−" sign is replaced by 1

To make positive no. $(+101.001)_2 = (0101.001)_2$
To make negative no. $(-101.101)_2 = (1101.101)_2$

⇒ There are two ways of -
(1) - Representation of signed numbers:-

(3) II. Complement form
I (DRC) Diminished Radix Complement
OR $(r-1)$'s Complement

II (RXC) - Radix Complement OR r's Complement

For example:-

⇒ Binary number
DRC is known as $(2-1)$ = One's complement (1's Complement)

RXC is known as 2's (two's) Complement

⇒ In Decimal number
DRC is known as 9's Complement
RXC is known as 10's Complement

QUALCOMM

One's Complement Representation

The most Significant bit (MSB) represents the Sign.

I If MSB is a "0"
The number is ~~zero~~ positive
Remaining (n-1) bits directly indicates the magnitude.

II If MSB is "1"
The number is negative
Complement of all the remaining (n-1) bits gives the magnitude

For example — ~~For~~ a binary representation $2^8 - 1 = 11111111$
11110011 ∵ First bit is 1 : The no. is negative
One's Complement of 11110011 → 00001100
→ $(12)_{10}$

Range of n-bit numbers —
One's Complement numbers :-

0111111	→	+63
0000110	→	+6
0000000	→	+0
1111111	→	+0
1111001	→	-63
1000000	→	-63

7 bit number covers the range from +63 to -63
n bit number has a range from $+(2^{n-1}-1)$ to $-(2^{n-1}-1)$

⇒ One's Complement of a number
Complement all the digits

- if A is an integer in one' Complement form then one's Complement of A = −A
- The applies to fractions as well

A = 0.101 $(+0.625)_{10}$

One's Complement of A = 1.010 $(-0.625)_{10}$

Mixed number

B = 010011.0101 $(+19.3125)_{10}$

One's Complement of B = 101100.1010 $(-19.3125)_{10}$

→ The representation of +51 and −51 in both 2's and 1's Complement forms is shown below :—

Sign Bit	Magnitude							
0	1	1	0	0	1	1		+51 (In sign magnitude form)
1	1	1	0	0	1	1		−51 (In Sign magnitude form)
1	0	0	1	1	0	1		−51 (In 2's Complement form)
1	0	0	1	1	0	0		−51 (In 1'S Complement form)

Question: Each of the following no's is a signed binary number. Determine the decimal value in each case, if they are in

	i) Sign magnitude form	ii) 2'S Complement	iii) 1'S Complement
01101	+ 13	+ 13	+ 13
010111	+ 23	+ 23	+ 23
10111	− 7	− 9	− 8
1101010	− 42	− 22	− 21

✓

Two's Complement Representation -

⇒ if MSD is a "0"
The no. is positive.
All remaining (n-1) bits directly indicates the magnitude.

⇒ If MSD is "1"
The number is negative.
Magnitude is obtained by complementing all the remaining (n-1) bits and adding a 1

For example:- 1111010 → (1) (111010)
First bit (sign) is 1 : The number is negative

Complement of 111010 is 000101
the add 1 $\quad + \quad 1$
$= 000110 = (6)_{10}$

Range of n bit numbers in 2's Complement
Two's Complement numbers:

0111111	→	+63	$2^{n-1}-1$
0000110	→	+6	↓
0000000	→	+0	0
1111010	→	-6	↓
1000001	→	-63	↓
1000000	→	-64	-2^{n-1}

Range

7-bit number covers the range from +63 to -64
n-bit number has a range from $+(2^{n-1}-1)$ to $-(2^{n-1})$

⇒ 2'S Complement of a number –
Complement all the digits and add '1' to the LSB
if A is an integer in One's Complement form, then, 2'S Complement of $A = -A$

This applies to fractions as well
- $A = 0.101$ $(+0.625)_{10}$
Two'S Complement of $A = 1.011 \rightarrow (-0.625)_{10}$

mixed number
- $B = 010011.0101$ $(+19.3125)_{10}$
- 2'S Complement of
$B = 101100.1011 \rightarrow (-19.3125)_{10}$

Characteristics of the 2'S Complement numbers:-
i) There is one unique zero.
ii) The 2'S Complement of 0 is 0
iii) MSB cannot be used to express a quantity. It is sign bit if it is zero, the no. is ~~negat~~positive & if it is 1, then the number is negative.
iv) For n-bit word which include the sign bit, there are $2^{n-1}-1$ positive integer, 2^{n-1} negative integer and one 0
v) Significant information is contained in the 1s of the positive nos and 0s of the negative numbers.
vi) A negative no. may be converted into a positive number by finding its 2'S Complement.

QUALCOMM

Methods of obtaining the 2'S Complement of a number :-

First method -

By obtaining the 1'S Complement of the given number (by changing all 0s to 1s and 1s to 0s) and then adding 1.

Second Method -

By substracting the given n-bit number N from 2^n

Third Method -

Starting at the LSB, copying down each bit up to and including the first 1 bit encountered, and complementing the remaining bits.

Que - Express -45 in 8-bit 2'S Complement form.

Ans - +45 in 8-bit form is 00101101

First method

Obtain 1'S Complement of 00101101 and then add 1

Positive expression of the given no. 00101101

1'S Complement of it 11010010

Add 1 + 1

11010011

Second Method

Substract the given no. N from 2^n

$2^n = 2^8 =$ 100000000

Substract 45 = 00101101

The the 2'S Complement of -45 is 01101011

Third method

Original number 00101101

Copy up to the first 1 bit 1

Complementing the remaining bits 1101001

2's Complement form of −45 is 11010011

Que. Express −73.75 in 12-bit 2's Complement form.

Solution +73.75 = N = 01001001.1100

Third method

Original number 01001001.1100

Copy up to the first 1 bit 100

Complementing the remaining bits 10110110.0

10110110.0100

* One's Complement Arithmetic

⇒ ∘ This system is very easy to implement in hardware by simply feeding all bits through inverters

∘ The main difficulties of 1's complement is its representation of zero.

Both 00000000 and its 1's complement 11111111 represent zero.

The 00000000 is called positive zero and the 11111111 is called negative zero.

⇒ ∘ In 1's Complement Substraction, add the 1's complement of the subtrahend to the minuend.

∘ If there is a carry out, bring the carry around and add it to the LSB. This is called end around carry.

∘ If MSB (sign bit) is 0, then the result is positive and is in true binary.

∘ If MSB is 1 (whether there is a carry or no carry at all), the result is negative and is in its

1's Complement form.
By taking 1's complement we get the magnitude in binary.

Example:- Substract 14 from 25 using 8-bit 1's complement

Solution:-

25	+14 = 00001110	
−14	−14 = 11110001	(in 1's Complement form)
+11	+25 = 00011001	

$$
\begin{array}{rl}
25 \Rightarrow & 00011001 \\
-14 \Rightarrow & 11110001 \\
\hline
+11 & 1\,00001010 \\
 & \quad\quad\quad\quad +1 \\
\hline
 & 00001011 = +(11)_{10}
\end{array}
$$

carry

Que Add −25 to −14 using the 8-bit 1's complement method

Solution +25 00011001

+14 00001110

1's Complement of +25 = −25

= 11100110

1's Complement of +14 = −14

= 11110001

$$
\begin{array}{rl}
-25 & 11100110 \quad \text{(in 1's Complement form)} \\
-14 \Rightarrow & +\,11110001 \quad (\text{—}\,\text{—}) \\
\hline
-39 & 1\,10010111 \\
 & \quad\quad\quad\quad +1 \quad \text{(Add the end around carry)} \\
\hline
 & 11011000
\end{array}
$$

The MSB is 1. So the result is negative and is in its 1's Complement form.
The 1's Complement of 11011000 is 00100111.
Therefore the result is −39.

Question:- Substract 27.50 from 68.75 using the 12-bit 1's complement arithmatic

Solution

+68.75	01000100.1100	(in 1's complent form)
−27.50	+ 11100100.0111	
+41.25	1 00101001.0011	
	+ 1	(Add end around carry)
	00101001.0100	

The MSB is 0, so it is positive no. and is in its normal binary form. Therefore the result is +41.25

Question- Add −89.75 to +43.25 using 12 bit 1's complement method.

Solution-

+43.25	00101011.0100
−89.75	+ 10100110.0011
−46.50	11010001.0111

There is no carry.

The MSB is 1, so it is negative number and the 1's complement of 11010001.0111 is 00101110.1000.

Therefore, the result is −46.50.

Two's Complement Arithmetic

- The 2's complement system is used to represent negative numbers using modulus arithmetic.
- The word length of a computer is fixed. if 4-bit number is added to another 4-bit number, the result will be only 4-bit.
- Carry from the fourth bit will overflow and it is ignored. This is called the modulus arithmetic.

In the 2's complement substraction,

- add the 2's complement of the subtrahend to the minuend. If there is any carry, ignore it.
- If the sign bit i.e MSB of the sum term is a 0, the result is positive and its in true binary form
- If the MSB is a 1, the result is negative and is in its 2's complement form.
- To find its magnitude in binary we have to take its 2's complement.

Que. Add −45.75 to +87.5 using 12-bit 2's complent arithmetic.

Solution:

+87.50	01010111.1000	
−45.75	+ 11010010.0100	(in 2's complement form)
+41.75	100100001.1100	(ignor the carry)

There is a carry, ignor it. The MSB is 0, so the result is positive and is in normal binary form. Therefore, the result is +41.75

Boolean Algebra :-

It may be defined with a set of elements, a set of operators and a no. of unproved axioms or postulates.

~~एक्सियम~~

A set of elements is any collection of objects having a common property.

If a set S has two elements x and y, then $x \in S$, denotes that x is a member of set S.

$y \notin S$, denotes that y is not an element of S.

A binary operator defined on a set of elements is a rule that assign to each pair of elements from S.

~~from~~ For example $a * b = c$ '*' binary operator

Axioms or postulates of boolean algebra are a set of logical expressions that we accept without proof and upon which it is possible to deduce the rules, theorems, and properties of the system.

Boolean Algebra is differs from both the ordinary algebra and the binary number system.

In Boolean Algebra, $A+A=A$, $A \cdot A = A$

Because variable A has only a logical value.

In ordinary Algebra, $A+A=2A$, $A \cdot A = A^2$

Because variable A has numerical value

In binary Number system, $1+1=10$

In boolean algebra $1+1=1$

A Boolean Function is an expression formed with binary variables, two binary operators OR & AND, the unary operator NOT, Parentheses and equal sign

Logical operations:-

The AND, OR and NOT are the three basic operation or functions that are performed in Boolean Algebra.
In addition, there are some derived operations such as NAND, NOR, Ex-OR and Ex-NOR are also preformed in Boolean algebra.

AND operation -

The AND operation in Boolean algebra is similar to multiplication in ordinary algebra. In fact, it is logical multiplication as performed by the AND gate.

$F = x \cdot y$ $\qquad$ $F = x$ AND y

OR operation -

The OR operation in Boolean algebra is similar to addition in ordinary algebra. In fact it is logical addition as performed by OR gate.

$F = x + y$ $\qquad$ $F = x$ OR y

NOT operation -

The NOT operation in Boolean algebra is nothing but complementation or inversion, that is negation as perfored by the NOT gate.
The NOT operation is indicated by a bar '—' over the variable.

$F = \overline{x}$ $\qquad$ $F =$ NOT x

NAND operation -

The NAND operation in Boolean algebra is equivalent to AND operation plus NOT operation i.e it is negation of AND operation as performed by the NAND gate. $F = \overline{x \cdot y}$ $\qquad$ NOT - AND

NOR-operation-

The NOR operation in Boolean algebra is equivalent to OR operation plus NOT operation i.e it is the negation of OR operation as performed by NOR gate

$F = \overline{x+y}$ NOT-OR

X-OR and X-NOR operation-

The X-OR and X-NOR operations on variable A and B in Boolean algebra are denoted by $A \oplus B (= \bar{A}B + A\bar{B})$ and $A \odot B (= AB + \bar{A}\bar{B}) \rightarrow$ A equals B Coincidence

A or B but not both

AXIOMS/ Postulate & Laws of Boolean Algebra

Postulate I. The operation are closed for all A and B $\in S$

i) $A+B \in S$ ii) $A \cdot B \in S$

Postulate II. i) $A+0 = A$ ii) $A \cdot 1 = A$

Postulate III cummutative i) $A+B = B+A$ ii) $AB = BA$

Postulate IV distributive i) $A(B+C) = AB+AC$

ii) $A+BC = (A+B)(A+C)$

Postulate V i) $A+A' = 1$ ii) $AA' = 0$

Theorem I i) $A+A = A$ ii) $A \cdot A = A$

Theorem II i) $A+1 = 1$ ii) $A \cdot 0 = 0$

Theorem III involution $(A')' = A$

Theorem IV associative i) $x+(y+z) = (x+y)+z$

ii) $x(yz) = (xy)z$

Theorem V De Morgan

i) $(x+y)' = x'y'$ ii) $(xy)' = x'+y'$

Theorem VI Absorption i) $x+xy = x$

ii) $x(x+y) = x$

Each Axiom can be interpreted as the outcome of an operation performed by logic gates (AND, OR, NOT)

Axiom: 1 $0 \cdot 0 = 0$

: 2 $0 \cdot 1 = 0$

: 3 $1 \cdot 0 = 0$

: 4 $1 \cdot 1 = 1$

: 5 $0+0 = 0$

Axiom: 6 $0+1 = 1$

: 7 $1+0 = 1$

: 8 $1+1 = 1$

: 9 $\bar{1} = 0$

: 10 $\bar{0} = 1$

I. Complementation Laws

The term complement means to invert i.e to change 0s to 1s and 1s to 0s.

The five laws of complementation are as follows:

Law 1: $\bar{0} = 1$

Law 2: $\bar{1} = 0$

Law 3: if $A = 0$ then $\bar{A} = 1$

Law 4: if $A = 1$ then $\bar{A} = 0$

Law 5: $\bar{\bar{A}} = A$

II AND laws

The four AND laws are as follows

Law 1: $A \cdot 0 = 0$

Law 2: $A \cdot 1 = A$

Law 3: $A \cdot A = A$

Law 4: $A \cdot \bar{A} = 0$

III OR laws The 4 OR laws are as follows

Law 1: $A+0 = A$

Law 2: $A+1 = 1$

Law 3: $A+A = A$

Law 4: $A+\bar{A} = 1$

QUALCOMM

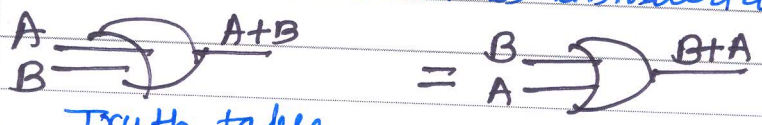
IV Commutative laws -

Commutative laws allow the change in position of AND or OR variables.

There are two Commutative laws -

Law 1: $A+B = B+A$ Law 2: $A \cdot B = B \cdot A$

LAW 1: This law states that, A OR B is the same as B OR A i.e the order in which the variables are ORed is immaterial.



Truth table

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

B	A	B+A
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table: A table which lists all the possible combinations of inputs variables and the corresponding outputs is called a truth table.

It shows how the logic circuit's output responds to various combinations of logic levels at the inputs.

This law can be extended for any no. of variables

Per exp- $A+B+C = B+C+A = C+A+B = B+A+C$

Law 2: $A \cdot B = B \cdot A$

This law states that, A AND B is the same as B AND A i.e the order in which the variables are ANDed is immaterial / not important.

It makes no difference, which input of an AND gate is connected to A and which to B.

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

=

B	A	B·A
0	0	0
0	1	0
1	0	0
1	1	1

This law can be extended to any no. of variables

$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$

V Associative Laws

This laws allows grouping of variables.

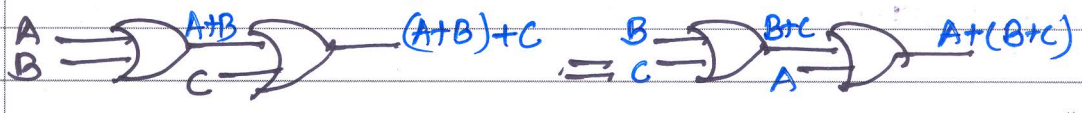
There are two associative laws.

Law 1: $(A+B)+C = A+(B+C)$

A OR B ORed with C is the same as A ORed with B OR C.

This law states that the way the variable are grouped and ORed is immaterial.

The truth tables given ~~next~~ below illustrate this law.



A	B	C	A+B	(A+B)+C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	B+C	A+(B+C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

This law can be extended to any number of variables.

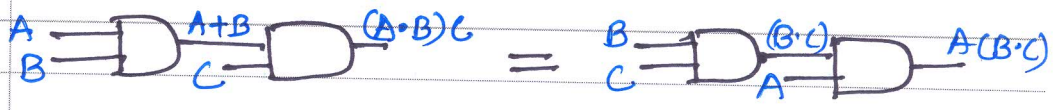
$A+(B+C+D) = (A+B+C)+D = (A+B)+(C+D)$

Law 2: $(A \cdot B)C = A(B \cdot C)$

A AND B ANDed with C is the same as A ANDed with B AND C.

The way the variables are grouped and ANDed is immaterial (unimportant, irrelevant)

The truth tables given below illustrate this law:-



A	B	C	AB	(AB)C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

A	B	C	BC	A(B·C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

QUALCOMM

This law can be extended to any number of variables
$A(BCD) = (ABC)D = (AB)(CD)$

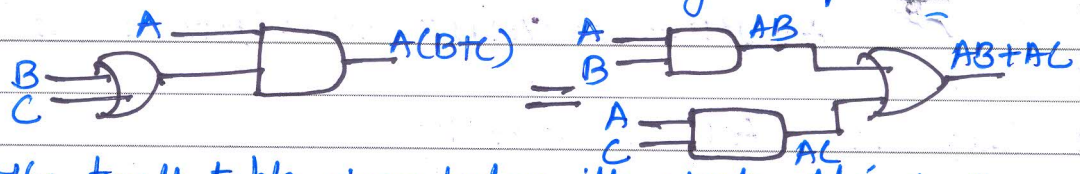
VI Distributive Laws :-

This law allows factoring or multiplying out of expressions.

There are three distributive laws -

Law 1: $A(B+C) = AB + AC$

This law states that ORing of several variables and ANDing the result with a single variable is equivalent to ANDing that single variable with each of the several variables and then ORing the products.



The truth table given below illustrate this law -

A	B	C	B+C	A(B+C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	AB	AC	AB+AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

The distributive property is often used in the reverse.

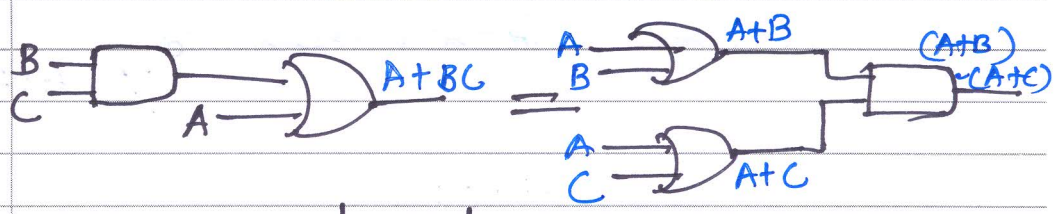
That is, $AB + AC = A(B+C)$

Law 2: $A + BC = (A+B)(A+C)$

This law states that ANDing of several variables and ORing the result with a single variable is equivalent to ORing that single variable with each of the several variables and then ANDing the sums.

This can be proved algebraically as -

$$
\begin{aligned}
R.H.S &= (A+B)(A+C) \\
&= AA + AC + BA + BC \\
&= A + AC + AB + BC && \{\because A \cdot A = A\} \\
&= A(1+C) + AB + BC \\
&= A \cdot 1 + AB + BC && \{\because 1+C = 1\} \\
&= A(1+B) + BC \\
&= A \cdot 1 + BC && \{\because 1+B = 1\} \\
&= A + BC \\
&= L.H.S
\end{aligned}
$$



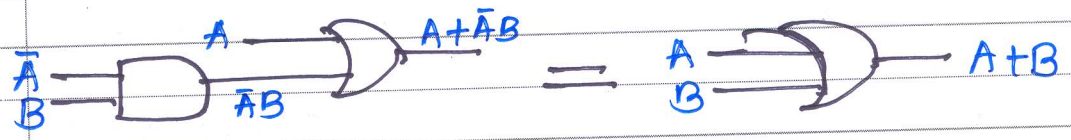
A	B	C	BC	A+BC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

=

A	B	C	A+B	A+C	(A+B)(A+C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Law 3: $A+\bar{A}B = A+B$

This law states that ORing of a variable with the AND of the complement of that variable with another variable, is equal to the ORing of this two variable.



The truth table illustrate this law –

A	B	$\bar{A}$	$\bar{A}B$	$A+\bar{A}B$
0	0	1	0	0
0	1	1	1	1
1	0	0	0	1
1	1	0	0	1

=

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

This law can proved algebraically as

$$\begin{aligned} L.H.S &= A+\bar{A}B \\ &= (A+\bar{A})(A+B) \\ &= 1\cdot(A+B) \qquad \{\because A+\bar{A}=1\} \\ &= A+B \\ &= R.H.S \end{aligned}$$

VII Idempotence Laws

Idempotence means the same value.

Law 1: $A\cdot A = A$

This law states that ANDing of a variables with itself is equal to that variable only.

if $A=0$, then $A\cdot A = 0\cdot 0 = 0 = A$

if $A=1$, then $A\cdot A = 1\cdot 1 = 1 = A$

Law 2: $A+A = A$ This law states that ORing of a variable with itself is equal to that variable only.

if $A=0$, then $A+A = 0+0 = 0 = A$

if $A=1$, then $A+A = 1+1 = 1 = A$

VIII Complementation Laws or Negation Laws —

There are two Complementation or negation laws.

Law 1 : $A \cdot \bar{A} = 0$

A variable ANDed with its Complement is always equal to zero.

That is, if $A = 0$, then $\bar{A} = 1$ and $A \cdot \bar{A} = 0 \cdot 1 = 0$

if $A = 1$, then $\bar{A} = 0$ and $A \cdot \bar{A} = 1 \cdot 0 = 0$

Law 2 : $A + \bar{A} = 1$

A variable ORed with its Complement is always equal to 1 One.

That is, if $A = 0$, then $\bar{A} = 1$ and $A + \bar{A} = 0 + 1 = 1$

if $A = 1$, then $\bar{A} = 0$ and $A + \bar{A} = 1 + 0 = 1$

(IX) Double negation law : $\bar{\bar{A}} = A$

This law states that the double negation of a variable is equal to the variable itself.

if $A = 0$, then $\bar{\bar{A}} = \bar{\bar{0}} = \bar{1} = 0 = A$

if $A = 1$, then $\bar{\bar{A}} = \bar{\bar{1}} = \bar{0} = 1 = A$

The odd number of negation is equivalent to a single inversion and any even no. of inversions is equivalent to no inversion or the variable itself

(X) Identity Laws

There are two identity law :-

Law 1 $A \cdot 1 = A$

This law states that a variable ANDed with 1 is equal to the variable itself. That is,

if $A = 0$, then $A \cdot 1 = 0 \cdot 1 = 0 = A$

if $A = 1$, then $A \cdot 1 = 1 \cdot 1 = 1 = A$

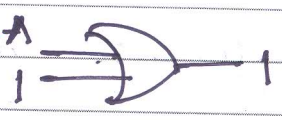
16-1-2018

Law 2 $A+1=1$

This law state that a variable ORed with 1 is always equal to 1.

if $A=0$, then $A+1=0+1=1$

if $A=1$, then $A+1=1+1=1$



(XI) Null Laws

Law 1 $A \cdot 0 = 0$

This law states that a variable ANDed with 0 is always equal to 0. That is,

if $A=0$, then $A \cdot 0 = 0 \cdot 0 = 0$

if $A=1$, then $A \cdot 0 = 1 \cdot 0 = 0$

Law 2 $A+0=A$

This law states that a variable ORed with 0 is equal to the variable itself. That is,

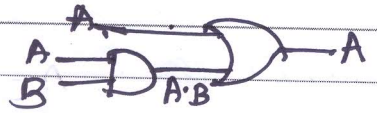
if $A=0$, then $A+0=0+0=0=A$

if $A=1$, then $A+0=1+0=1=A$

(XII) Absorption Laws

Law 1: $A+A \cdot B = A$

This law states that ORing of a variable with ANDed of that variable with another variable is equal to that variable itself. → Truth table



A	B	A B	(A+A·B)
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Algebraically,

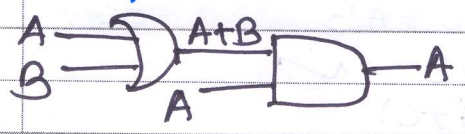
$A+A \cdot B = A(A+B)$

$= A \cdot 1 = A$

QUALCOMM

Law 2. $A(A+B) = A$

This law states that a variable is ANDed with the OR of that variable and another variable is equal to that variable itself.



Truth table

A	B	(A+B)	A·(A+B)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Algebraically,

$A(A+B) = A \cdot A + AB$

$= A + AB$

$= A(1+B)$

$= A \cdot 1 = A$

THEORGM :—

I. (~~XIII~~) Consensus Theorem (Included factal theorem)

Theorem 1 : $AB + \bar{A}C + BC = AB + \bar{A}C$

proof. $L.H.S = AB + \bar{A}C + BC$

$= AB + \bar{A}C + BC \cdot 1$

$= AB + \bar{A}C + BC(A + \bar{A})$

$= AB + \bar{A}C + BCA + BC\bar{A}$

$= AB + \bar{A}C + ABC + \bar{A}BC$

$= AB + ABC + \bar{A}C + \bar{A}BC$

$= AB(1+C) + \bar{A}C(1+B)$

$= AB \cdot 1 + \bar{A}C \cdot 1$

$= AB + \bar{A}C$

$= R.H.S$

This theorem can be extended to any number of variables.

Theorem 2:

$$(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$$

Proof: L.H.S $= (A+B)(\bar{A}+C)(B+C)$

$= (A+B)(C+\bar{A}B)$: (Distributive law)

$= AC + A\cdot\bar{A}B + BC + B\bar{A}B$

$= AC + 0 + BC + \bar{A}B$ $\quad A\cdot\bar{A}=0$, $B\cdot B = B$

$= AC + BC + \bar{A}B$ ✓

R.H.S $= (A+B)(\bar{A}+C)$

$= A\cdot\bar{A} + AC + B\bar{A} + BC$

$= 0 + AC + BC + \bar{A}B$ $\quad \{\because A\cdot\bar{A}=0\}$

$= AC + BC + \bar{A}B$ ✓

$=$ L.H.S

II. Transposition Theorem

$$AB + \bar{A}C = (A+C)(\bar{A}+B)$$

R.H.S $= (A+C)(\bar{A}+B)$

$= A\cdot\bar{A} + AB + C\bar{A} + CB$

$= 0 + AB + \bar{A}C + BC\cdot 1$

$= AB + \bar{A}C + BC(A+\bar{A})$

$= AB + \bar{A}C + ABC + \bar{A}BC$

$= AB + ABC + \bar{A}C + \bar{A}CB$

$= AB(1+C) + \bar{A}C(1+B)$

$= AB\cdot 1 + \bar{A}C\cdot 1$

$= AB + \bar{A}C$ $= $ L.H.S

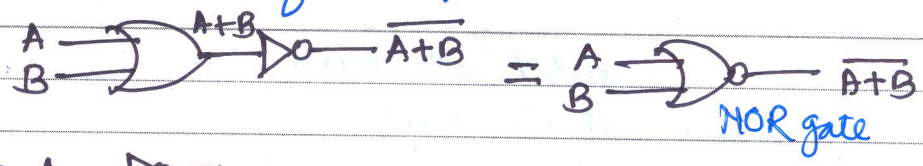
III. De-Morgan's Theorem –

De-morgan's theorem represents the two most powerful laws in Boolean algebra.

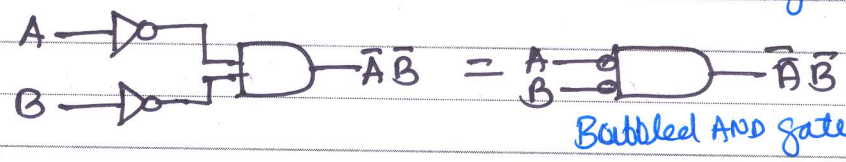
Law 1: $\overline{A+B} = \bar{A}\cdot\bar{B}$

This law states that complement of ORed of two or more variable is equal to the AND of the complements of each of the individual variables.

L.H.S =



R.H.S =



A	B	A+B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

=

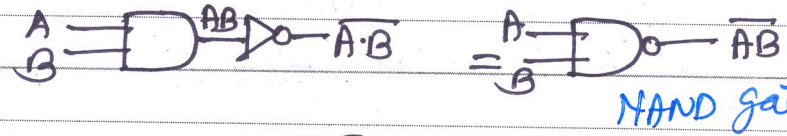
A	B	$\bar{A}$	$\bar{B}$	$\bar{A}\cdot\bar{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

It shows that the NOR gate is equivalent to a bubbled AND gate.

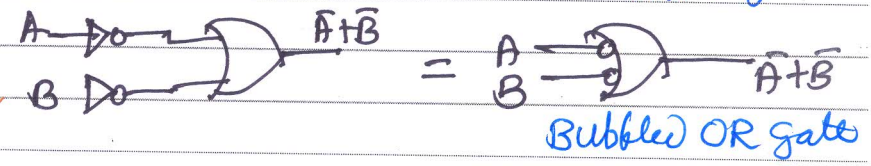
Law 2: $\overline{A\cdot B} = \bar{A}+\bar{B}$

This law states that complement of ANDed two or more variable is same as ORed of the complements of each of the individual variables.

L.H.S =



R.H.S =



Truth Table –

A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

≡

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}+\bar{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

It shows that NAND gate is equivalent to bubbled OR gate.

Demorgarization procedure is as follows –

1. Complement the entire given function
2. Change all the ANDs to ORs and all the ORs to ANDs.
3. Complement each of the individual variables.

Ex:– Demorganize $\overline{(A+\bar{B})(C+\bar{D})}$

Soln:– The given function is $\overline{(A+\bar{B})(C+\bar{D})}$

Step 1. Complement the entire function

$$\overline{\overline{(A+\bar{B})(C+\bar{D})}} = (A+\bar{B})(C+\bar{D})$$

Step 2. Change ANDs to ORs and ORs to ANDs

$$A\bar{B}+C\bar{D}$$

Step 3. Complement each variable $\bar{A}\bar{\bar{B}}+\bar{C}\bar{\bar{D}}$

$$(\bar{A}B+\bar{C}D)$$

Ex – Apply Demorgan's theorem to the expression

$$\overline{\overline{AB}(CD+\bar{E}F)(\overline{AB}+\overline{CD})}$$

Soln :–

$$= \overline{\overline{AB}} + \overline{(CD+\bar{E}F)} + \overline{(\overline{AB}+\overline{CD})}$$

$$= AB + \overline{CD}\cdot\overline{\bar{E}F} + \overline{\overline{AB}}\cdot\overline{\overline{CD}}$$

$$= AB + (\bar{C}+\bar{D})(\bar{\bar{E}}+\bar{F}) + \cancel{(\bar{A}+\bar{B})CD}\ ABCD$$

$$= AB + ABCD + (\bar{C}+\bar{D})(E+\bar{F})$$

QUALCOMM

DUALITY –

डिड्यूसेबल
निष्कर्ष निकालने योग्य

It states that every algebraic expression deducible (= answered or decided by thinking carefully about the known facts) from the axioms/postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. This property of Boolean algebra is called duality principle.

if ~~we~~ the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's

	Given expression	Dual
1.	$\bar{0} = 1$	$\bar{1} = 0$
2.	$0 \cdot 1 = 0$	$1 + 0 = 1$
3	$0 \cdot 0 = 0$	$1 + 1 = 1$
4.	$1 \cdot 1 = 1$	$0 + 0 = 0$
5.	$A \cdot 0 = 0$	$A + 1 = 1$
6.	$A \cdot 1 = A$	$A + 0 = A$
7.	$A \cdot A = A$	$A + A = A$
8.	$A \cdot \bar{A} = 0$	$A + \bar{A} = 1$
9.	$A \cdot B = B \cdot A$	$A + B = B + A$
10.	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
11	$A \cdot (B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
12	$A(A + B) = A$	$A + AB = A$
13	$A(A + B) = A \cdot B$	$A + A + B = A + B$
14	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A} \cdot \bar{B}$
15	$(A + B)(\bar{A} + C)(B + C) = A + B(\bar{A} + C)$	$AB + \bar{A}C + BC = A \cdot B + \bar{A}C$
16	$A + \bar{B}C = (A + \bar{B})(A + C)$	$A(\bar{B} + C) = A\bar{B} + AC$
17	$A + B = AB + AB + A\bar{B}$	$AB = (A + B)(\bar{A} + B)(A + \bar{B})$
18	$(A + B)(C + D) = AC + AD + BC + BD$	$AB + CD = (A + C)(A + D)(B + C)(B + D)$
~~19~~		
~~20~~		

QUALCOMM

Reducing Boolean Expressions –

The procedure is –

Step 1. Multiply all variables necessary to remove parentheses.

Step 2. if there is ~~one~~ two or more identical terms then only one term be retained and rest all terms dropped / ignored.

For example:– $AB + AB + AB = AB$

Step 3 if there will be a variable and its negation present in the same term then this term can be dropped.

For example– $A \cdot B\bar{B} = A \cdot 0 = 0$, $ACB\bar{C} = ABC\bar{C} = AB \cdot 0 = 0$

Step 4. if ~~for~~ pairs of terms are identical except for one variable then the larger term can be ignored

For example– $AB\bar{C}\bar{D} + AB\bar{C} = AB\bar{C}(\bar{D}+1) = AB\bar{C}$

Step 5. The pairs of terms which have the same variables with one are more variable complemented.

if ~~a~~ variable in one term of such a pair is complemented while in the second term is not. then such terms can be combined into a single term with that variable dropped.

For example – $AB\bar{C}\bar{D} + AB\bar{C}D$

$AB\bar{C}(\bar{D}+\bar{D}) = AB\bar{C} \cdot 1 = AB\bar{C}$

(ii) $AB(C+D) + AB(\overline{C+D})$

$AB[(C+D) + \overline{(C+D)}] = AB(x+\bar{x}) = AB \cdot 1 = AB$

Example: Reduce the expression $A[B + \bar{C}(\overline{AB + A\bar{C}})]$

$A[B + \bar{C}(\overline{AB} \cdot \overline{A\bar{C}})]$

$A[B + \bar{C}(\bar{A}+\bar{B})(\bar{A}+\bar{\bar{C}})]$

$AB + A\bar{C}(\bar{A}+\bar{B})(\bar{A}+C)$

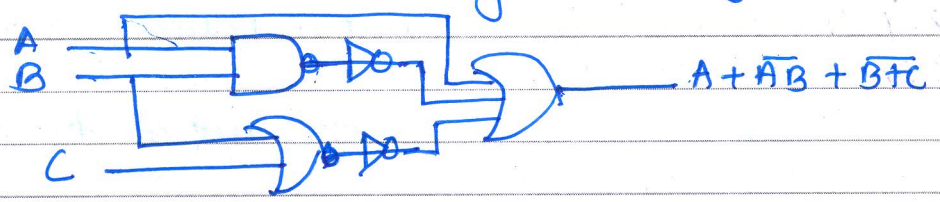
$AB + A\bar{C}(\bar{A}+\bar{B}C) = AB + A\bar{C}\bar{A} + A\bar{C}\bar{B}C$

$AB + 0 + 0 = AB$

QUALCOMM

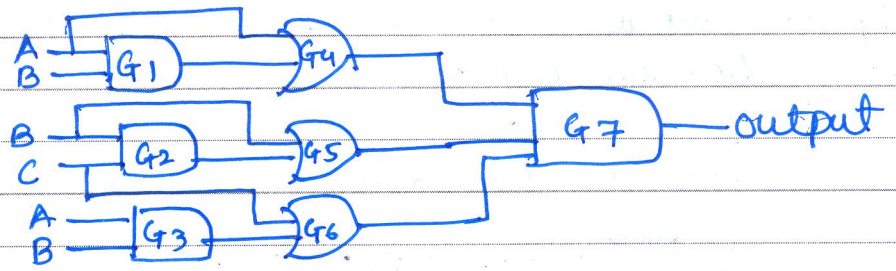
Converting Boolean Expressions to logic

Ex:- $\overline{A}B + A + \overline{B+C}$ using AOI logic



Converting logic to Boolean Expression

Ex:-



Soln:-

Starting from the input side and writing the expressions for the outputs of the individual gates output of different gates are -

$G_1 = AB$, $G_2 = BC$, $G_3 = AB$, $G_4 = A + G_1 = A + AB$

$G_5 = B + G_2 = B + BC$, $G_6 = C + G_3 = C + AB$

$G_7 = G_4 \cdot G_5 \cdot G_6 = (A+AB)(B+BC)(C+AB)$

$G_7 = A(1+B)\,B(1+C)(C+AB)$

$= A \cdot 1 \cdot B \cdot 1\,(C+AB) = AB(C+AB)$

$= ABC + AB \cdot AB = ABC + AB = AB(C+1)$

$= AB \cdot 1 = AB$

Converting AND/OR/Invert logic To NAND/NOR logic

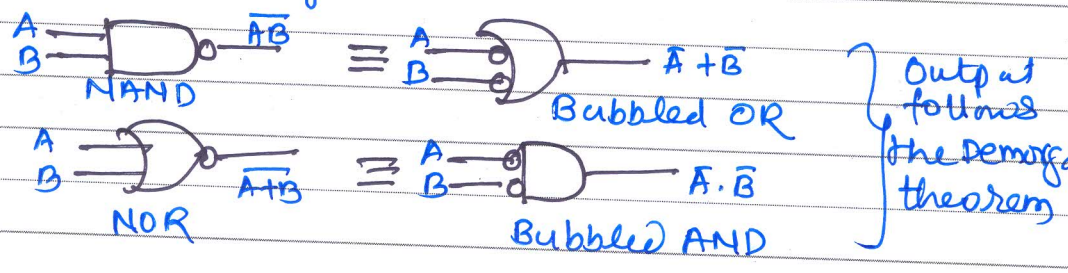
The procedure is :-

Step 1. Draw the circuit/expression in AOI logic

Step 2. if NAND logic is chosen, add a circle at the output of AND gate and at the inputs to all the OR gates

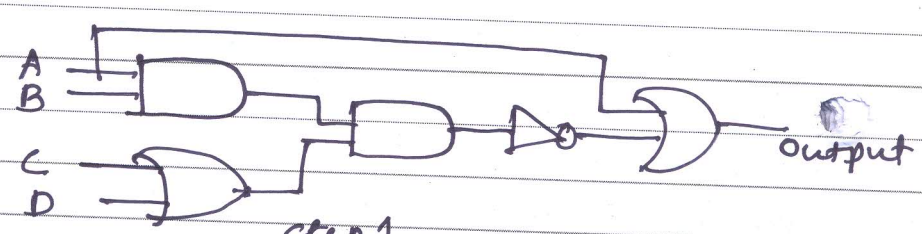
Step 2. if NOR logic is chosen, add a circle at the output of OR gate and at the inputs to all the AND gates.

Step 3. Add or subtract an inverter on each line that the polarity of the signals on those lines remains unchanged from that of the original diagram.

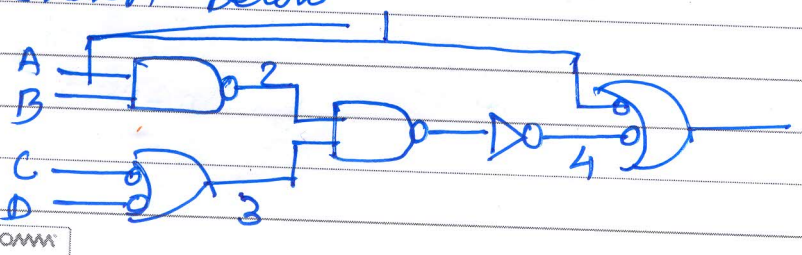


Ex:- Convert the following AOI logic circuit to (a) NAND logic (b) NOR logic

~~Soln:~~



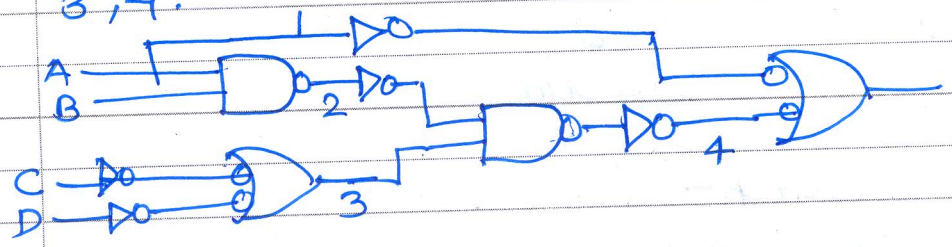
Soln:- (a) NAND logic - step 1 putting a circle at the o/p of each AND gate and at the input to all OR gates as shown below



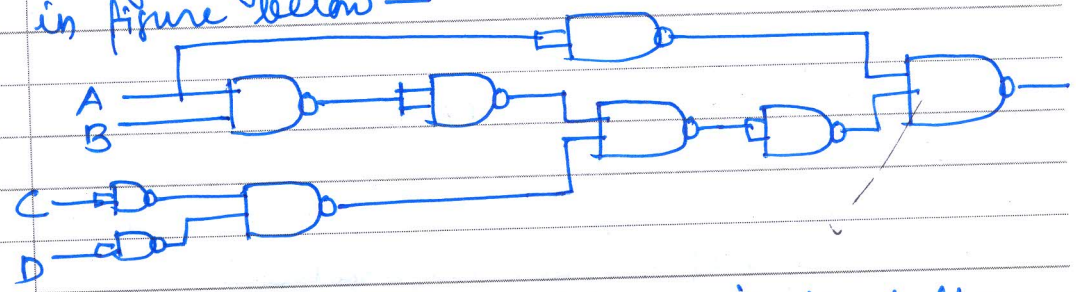
QUALCOMM

Step 2. Adding a inverter to each of the lines that received only one circle/bubble in the step 1. so that the polarity of these lines remains unchanged.

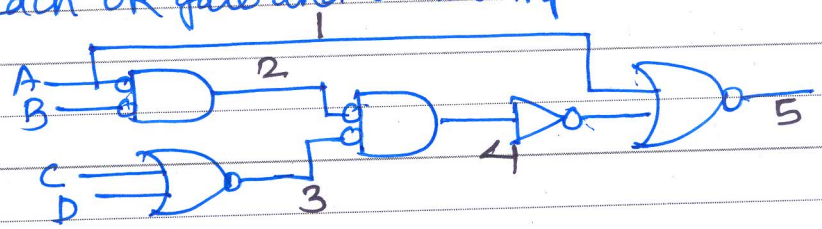
Inverters required in line C, D, 1, 2, Not in 3, 4.



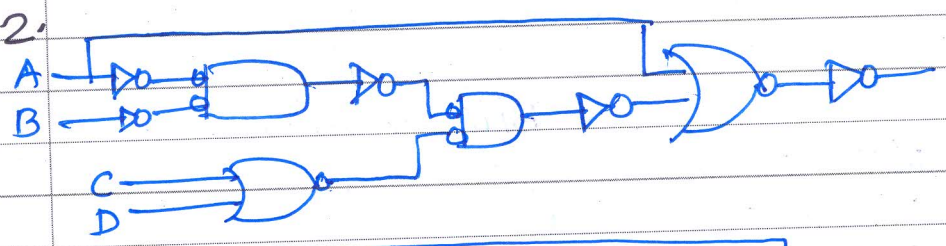
Using only NAND gates, the logic circuit as shown in figure below —



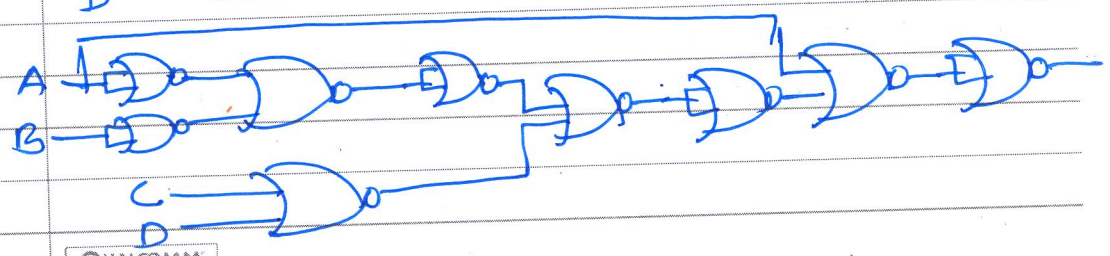
(b) NOR logic – step 1. putting a circle at the o/p of each OR gate and at the inputs to all AND gate



Step 2.



Step 3



QUALCOMM

Binary Codes

Need of coding –

Information sent over a noisy channel is likely to be distorted.

Information is coded to facilitate –

1. Efficient transmission
2. Error detection
3. Error correction

Coding – / Encoding

Coding ~~of~~ is the process of altering the characteristics of information to make it more suitable for intended (deliberate, intentional) application

Coding schemes depend on –

1. Security requirements
2. Complexity of the medium of transmission
3. Levels of error tolerated.
4. Need for standardization

Decoding –

- Decoding ~~of~~ is the process of reconstructing source information from the received encoded information
- Decoding can be more complex than coding if there is no prior knowledge of coding schemes.

Bit Combinations

Bit – a binary digit 0 or 1

Nibble – a group of four bits

Byte – a group of eight bits

word – a group of sixteen bits

(Some times used to designate 32 bit or 64 bit groups of bits)

QUALCOMM

Binary codes can be classified as numeric codes and alphanumeric codes.

Alphanumeric codes - It represents the alphanumeric information which is the combination of letters of the alphabet and decimal numbers as a sequence of 0s and 1s.

Numeric codes - It represents numeric information i.e only numbers as a series of 0s and 1s.

Numeric codes used to represent decimal digits are called Binary coded Decimal BCD codes.

In a BCD code the each digit of a decimal no. is encoded into groups of 4 binary digits.

A sequence of binary digits which represents a decimal digit is called a code word.

Disadvantages -

- BCD codes require more hardware.
- It slows down the system.
- There can be 30×10^{10} ($^{16}C_{10} \cdot 10!$) possible codes.
- Most of these codes will not have any special properties.

Example -

Natural BCD

Consider the number $(16.85)_{10}$

$(16.85)_{10} = (0001\,0110 \cdot 1000\,0101)$ NBCD

NBCD code is used in calculators.

QUALCOMM

Based upon the following properties we can select a coding scheme —

1. It should have ease of coding.
2. ease of arithmetic operations
3. minimum use of hardware
4. error detection property
5. ability to prevent wrong output during transistions.

The BCD code may be weighted or Non-weighted.

Weighted codes — It obeys the position weighting principle. each position of the number represents a specific ~~code~~ weight.

For ex:- $(A)_{10} = w_3 a_3 + w_2 a_2 + w_1 a_1 + w_0 a_0$

where Decimal no. $(A)_{10}$ is encoded in the binary form as $a_3 a_2 a_1 a_0$ and w_3, w_2, w_1 and w_0 are the weights selected for a given code.

The weighted code may be either positively-weighted or negatively-weighted.

Positively-weighted codes are those in which all weights assigned to the binary digits are positive. In every positively-weighted code, the first weight must be 1, the second weight must be either 1 or 2 and the sum of all the weights must be equal to or greater than 9.

Ex:-

w_3	w_2	w_1	w_0
8	4	2	1
2	4	2	1
5	2	1	1
3	3	2	1
4	3	1	1

In negatively weighted codes, the sum of weights assigned to the binary digits must be negative.

Ex:—

w_3	w_2	w_1	w_0
6	4	2	-3
6	3	1	-1
8	4	-2	-1
7	4	-2	-1

Non-weighted codes are codes, which do not obey the position-weighting ~~codes~~ principle.

Ex:— Excess-3 (XS-3) and Gray code.

Table – B.C.D codes

NW Codes Decimal digit	XS-3	Positively weighted code 8421	2421	5211	5421	Negatively weighted codes 642-3	84-2-1
0	0011	0000	0000	0000	0000	0000	0000
1	0100	0001	0001	0001	0001	0101	0111
2	0101	0010	0010	0011	0010	0010	0110
3	0110	0011	0011	0101	0011	1001	0101
4	0111	0100	0100	0111	0100	0100	0100
5	1000	0101	1011	1000	1000	1011	1011
6	1001	0110	1100	1000	1001	0110	1010
7	1010	0111	1101	1100	1010	1101	1001
8	1011	1000	1110	1110	1011	1010	1000
9	1100	1001	1111	1111	1100	1111	1111

There are sixteen four-bit groups.
10 of them is used to represent BCD codes
The rest unused six combinations are illegal
They may be utilised for error detection purposes.

QUALCOMM

Sequential Codes –

In a sequential code each succeeding code word is one binary number greater than its preceding code word.

Example:– 8421 and XS-3

The codes 5211, 2421 and 642-3 are not sequential.

Self-Complementing Codes –

Logical complement of a coded number is also its arithmetic complement.

A code is said to be self-complementing if code word of the 9's complement of N [i.e (9-N)] can be obtained from the code word of N by interchanging all the 0's and 1s.

In self complementing code, the code for 9 is the complement of the code for 0, 8 for 1, 7 for 2, 6 for 3, 5 for 4.

Ex:– 2421, 5211, 642-3, 84-2-1 and XS-3

The code 8421 and 5421 are not self complementing code.

- There are 4 (2421, 5211, 3321, 4311) positively-weighted ~~code~~ self complementing code.
- There are 13 negatively-weighted self codes.

Ex:– 2421 code

Nine's complement of $(4)_{10} = (5)_{10} = (9-4)_{10}$

2421 code for $(4)_{10} = (0100)$

Complement of 0100 = 1011 = 2421 code for $= (9-4)_{10} = (5)_{10}$

Necessary condition –

Sum of its weights should be 9.

QUALCOMM

Cyclic Codes – / Unit Distance Codes

In this codes each successive code word differs from the preceding one in only one bit position.

OR Adjecent codes differ only in one bit. They are also called unit distance codes.

Ex:– Gray Code. It is often used for translating an analog quantity such as shaft position into a digital form.

Appli:– for error detection & correction. Hocquenghem BCH

Decimal Digit	UDC-1	UDC-2	UDC-3
0	0000	0000	0000
1	0100	0001	1000
2	1100	0011	1001
3	1000	0010	0001
4	1001	0110	0011
5	1011	1110	0111
6	1111	1111	1111
7	0111	1101	1011
8	0011	1100	1010
9	0001	0100	0010

Reflective code – Imaged about the centre entries with one bit changed.

Ex:– 9'S Complement of a reflected BCD code word is formed by changing only one of its bits.

Decimal digit	Code-A	Code-B
0	0000	0100
1	0001	1010
2	0010	1000
3	0011	1110
4	0100	0000
5	1100	0001
6	1011	1111
7	1010	1001

Decimal digit	Reflected BCD code Code-A	Code-B
8	1001	1011
9	1000	0101

GRAY CODE – It is non-weighted code / cyclic code / reflective code and it is not suitable for arithmetic operations. It is not a BCD code.

- The n least significant bits (LSB) for 2^n through $(2^{n+1}-1)$ are the mirror images of those for 0 through (2^n-1).
- An N-bit gray code can be obtained by reflecting an (N-1) bit code about an axis at the end of the code, and putting MSB of 0 above the axis and MSB of 1 below the axis.

Constructing Gray code –

- The bits of gray code words are numbered from right to left from 0 to n-1.
- Bit i is 0 if bits i and (i+1) of the corresponding binary code word are the same, else bit i is 1
- When i+1=n, bit n of the binary code word is considered to be 0.

Ex:– Consider the decimal no. 68

$(68)_{10} = (1000100)_2$

Binary code : 1000100

Gray code : 1100110

- Gray code can easily converted to and from binary.
- Gray codes are used in instrumentation and data acquisition system where linear or angular displacement is measured.
- They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

Reflection of Gray Codes –

Decimal Digit	1 bit ~~Binary~~	1 bit gray	2 bit Binary	2 bit gray	3 bit Binary	3 bit Gray	4 bit Binary	4 bit Gray
0	0	0	0 0	0 0	000	000	0000	0000
1	1	1	0 1	0 1	001	001	0001	0001
2			1 0	1 1	010	011	0010	0011
3			1 1	1 0	011	010	0011	0010
4					100	110	0100	0110
5					101	111	0101	0111
6					110	101	0110	0101
7					111	100	0111	0100
8							1000	1100
9							1001	1101
10							1010	1111
11							1011	1110
12							1100	1010
13							1101	1011
14							1110	1001
15							1111	1000

Position Indicating System :– / Shaft encoder

A shaft encoder consists of a disk in which concentric circle have alternate sectors with reflective surfaces while the other sectors have non-reflective surfaces.

Binary to Gray Conversion —

if an n bit binary number is represented by $B_n B_{n-1} \cdots B_2 B_1$ and it Gray code equivalend by $G_n G_{n-1} \cdots G_2 G_1$ where B_n and G_n are MSBs the the gray code bits are obtained from the binary code as follows.

$G_n = B_n$

$G_{n-1} = B_n \oplus B_{n-1}$

$G_{n-2} = B_{n-1} \oplus B_{n-2}$

$\vdots$

$G_1 = B_2 \oplus B_1$

Symbol $\oplus$ stands for Exclusive OR (XOR)

The conversion procedure is as follows:

1. write the MSB of the binary as the MSB of the Gray code.
2. Add the MSB of the binary to the next bit in binary, writing/recording the sum and ignoring the carry, if any, i.e XOR the bits. This sum is the next bit of the gray code.
3. Add 2nd bit of binary to the 3rd bit of binary, the 3rd bit to the 4 bit and so on.
4. write the successive sums as the successive bits of the gray code until all the bits of the binary numer are exhausted/used.

second Method / shifting method :-

1. Exclusive OR the bits of binary no. with those of the binary number shifted one position to the right.
LSB of the shifted no. is discarded and the MSB of the Gray Code no. is the Same as the MSB of the original binary number.

QUALCOMM

Que. Convert the Binary 1001 to gray code

Soln:- (a)

Binary	1	0	0	1
	↓	↓	↓	↓
Gray	1	1	0	1

(b)

Binary		1	0	0	1
shifted binary	↓	1	0	0	1 — discard
Gray	1	1	0	1	

Gray to Binary Conversion —

if an n-bit gray no. is represented by $G_n G_{n-1} \cdots G_2 G_1$ and its binary equivalent by $B_n B_{n-1} \cdots B_2 B_1$, then binary bits are obtained from Gray bits as follows:

$$B_n = G_n$$
$$B_{n-1} = B_n \oplus G_{n-1}$$
$$B_{n-2} = B_{n-1} \oplus G_{n-2}$$
$$\vdots$$
$$B_2 = B_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1 = B_2 \oplus G_1$$

The conversion procedure is:

1. MSB of the binary no. is the same as the MSB of the Gray code number.
2. Add the MSB of the binary no. to the next significant bit of the Gray code i.e XOR them write the sum and ignor the carry.
3. Continue this till the all the gray bits are converted into binary

Que: Convert the Gray code 1101 into Binary code

Soln:- Conversion is done as shown below:-

Gray	1	1	0	1
	↓ ⊕	↓ ⊕	↓ ⊕	↓
Binary	1	0	0	1

QUALCOMM

The 8421 BCD Code –

In this code, each decimal digit, 0 through 9, is coded by a 4-bit binary number.

It is also called natural binary code because of the 8,4,2,1 weights attached to it.

It is a weighted code and is also sequential. Hence, it is useful for mathematical operations.

Advantage – ease of conversion to and from decimal.

Disadvantages – 1. It is less efficient than pure binary, it require more bits to represent a decimal no.

ex:– The decimal no. 14 can be represented in Pure Binary is 1110 but in 8421 BCD code is 00010100.

2. Arithmetic operations are more complex than they are in pure binary.

3. There are six illegal combination 1010, 1011, 1100, 1101, 1110, 1111 in this code.

BCD arithmetic –

The rules of binary addition and substration do not apply to the entire 8421 number but only to the individual 4-bit groups.

If there is a carry out of one group to the next group, or if the result is an illegal code, then 6_{10} (0110) is added to the sum term of that group and the resulting carry is added to the next group.

BCD Substration – It is performed by

Substracting the digits of each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend.

If there is a borrow from the next group ~~of the minuend~~ then 6_{10} (0110) is substracted from the difference term of this group.

In practice, substraction is performed by the complement method.

Example:- Addition in 8421 BCD code.

679.6 + 536.8

Soln:-

```
679.6  ⇒ 0110 0111 1001.0110   (In BCD)
+ 536.8  +0101+0011+0110+1000   (In BCD)
1216.4   1011 1010 1111.1110   (All are illegal codes)
         +0110+0110+0110+0110   (Add 0110 to each)
         1000 1000 0101 1.0100  (propagat carry)
          +1   +1   +1   +1
         0001 0010 0001 0110.0100  (corrected sum)
          1    2    1    6 . 4
```

Ex:- 25 + 13

Soln:-

```
  25        0010 0101   (In BCD)
+ 13   ⇒  +0001 0011   (  "  )
  38        0011 1000   (No carry, No illegal code. So this is the correct sum)
                3   8
```

Q. Binary Substration in 8421 BCD

(i) 38 − 15

Soln:-

```
 38        0011 1000
-15   =>  +0001 0101
----      ----------
 23        0100 1101   (No Borrow, So this is
                        Correct difference)
```

(ii) 206·7 − 147·8

Soln:-

```
 206·7        0010 0000 0110 · 0111   {In BCD}
-147·8   =>  -0001 0100 0111 · 1000
-----        ----------------------
 058·9        0000 1011 1000 · 1111   (Borrow are present so subtract 0110)
                  -0110 -0110 -0110
              ----------------------
              0000 0101 1000 · 1001
                 0    5    8   ·  9   Correcter difference
```

Q. Substration in BCD in 9'S Complement method

(i) 85 − 24

Soln:-

```
 85          85          (∵ 99−24=75)
-24   =>  + 75          (9'S Complement of 24)
 61        ---
           (1)60
             + 1        (End arrerund carry)
             ---
             61

 85₁₀        1000 0101   (In BCD)
+ 75₁₀   =>  + 0111 0101   (9'S Complement of 24 In BCD)
 ---         ------------
 ~~0~~       1111 1010   (Both are illegal Codes So add 0110 to each)
            +0110+0110
            ----------
            (1)0101 0000
            +1   +1
            ----------
            1   0110 0000
                       +1   (End arround carry)
            ----------
            0110 0001   (correcter difference)
            ----------
                 6  1
```

(ii) 305.5 − 168.8

9's Complement of 168.8 is 999.9 − 168.8 = 831.1

305.5 ⇒ 0011 0000 0101 · 0101
+831.1 +1000 +0011 +0001 +0.0001

1011 0011 0110 · 0100 (1011 is an illegal code)
+0110

1 0001 0011 0110 · 0100
+1 end around carry

0001 0011 0110 · 0111 (correct difference)

(iii) 679.6 − 885.9 = −206.3

Soln:— 9's Complement of 885.9 is 999.9 − 885.9 = 114.0

679.6 0110 0111 1001 · 0110 (In BCD
+114.0 ⇒ 0001 0001 0100 · 0000 (9's complement of 114.0 in BCD)
NO carry 793.6

0111 1000 1101 · 0110 (1101 is illegal so add 0110)
+0110

0000 1000 0011 · 0110 (Propagate carry)
+1

0000 1000 0011 · 0110
7 8 9 3 . 6 (793.6 in BCD)

No carry, so the result is negative and it is in 9's complement form.

9's Complement of 793.6 is 999.9 − 793.6 = 206.3

So the result is −206.3.

Q. Subtract in 8421 code using 10's Complement method

Soln:- (i) 342.7 − 108.9

10's Complement of 108.9 is ~~(999.9−108.9+1)~~ =

1000.0 − 108.9 = 891.1

```
  342.7          342.7
- 108.9        + 891.1   (10's complement of 108.9)
  233.8         (1)233.8  (signal carry)
```

```
342.7     0011 0100 0010 . 0111
+891.1  + 1000 1001 0001 . 0001   (10's complement of 108.9 in BCD)
          1011 1101 0011 . 1000   (1011, 1101 is illegal)
        + 0110 + 0110
        (1)0001 (1)0011 0011 . 1000   (propagate carry)
          +1     +1
        (1) 0010 0011 0011 . 1000   (signal carry)
            0010 0011 0011 . 1000   (Corrected difference)
            2    3    3    . 8
```

There is carry so result is positive.

(ii) 206.4 − 507.6

Soln:-

```
  206.4          206.4
- 507.6     ⇒   492.4   (10's Complement of 507.6)
- 301.2         698.8   (No Carry)
```

No carry so the result is negative and is 10's complement form.

The 10's complement of 698.8 is

1000.0 − 698.8 = 301.2

So the result is − 301.2.

206·4	0010	0000	0110	·0100	(206·4 in BCD)
+ 492·4	0100	1001	0010	·0100	(10's complement of 507·6 in BCD)
+698·8	0110	1001	1000	·1000	(No carry)
	6	9	8	·8	

So the result is −301·2

The Excess-3 code (XS-3)

It is non weighted BCD code. In this code each binary code word is the corresponding 8421 code word plus 0011 (3).

It is sequential code and can be used for arithmetic operations.

It is self complementing code. Substraction by complement addition method is more direct in XS-3 code than that in 8421 code.

XS-3 code has six invalid states 0000 (0), 0001 (1), 0010 (2), 1101 (13), 1110 (14), and 1111 (15)

XS-3 Arithmetic

Addition in XS-3 code —

1. We add XS-3 numbers by adding the 4-bit groups in each column starting from LSD.
2. If there is no carry out from addition of any of the 4-bit groups, substract 0011 from the sum term of those groups.
   [Because when 2 decimal digits are added in XS-3 and there is no carry, the result is in XS-6]
3. If there is a carry out, add 0011 to the sum term of those groups.
   [Because when there is carry, the invalid states are skipped and the result is in normal binary not in XS-3 code]

Substraction in XS-3 code –

1. Substract the XS-3 nos by substracting each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend starting from LSB.
2. If there is no borrow from the next 4-bit group add 0011 to the difference term of such groups
(Because when decimal digits are substracted in XS-3 and there is no borrow, the result is in normal binary not XS-3)
3. If there is a borrow, substract 0011 from the difference term.
(Because taking borrow is equivalent to adding six invalid states, so the result is in (XS-6))

In practice, substraction is performed by the 9'S complement or 10'S complement method

Q. Addition in XS-3 code

(i) 5+3

Soln:–

$$\begin{array}{r} 5 \\ +\ 3 \\ \hline 8 \end{array} \Rightarrow \begin{array}{rl} 1000 & \text{(In XS-3)} \\ +\ 0110 & \text{(In XS-3)} \\ \hline 1110 & \text{(NO carry)} \\ -\ 0011 & \text{(Subtract 0011)} \\ \hline 1011 & \text{(Corrected sum in XS-3)} \end{array}$$

★ Summary –

1. NO carry – Substract 0011
   Carry generated – add 0011

2. NO Borrow – add 0011
   Borrow taken – substract 0011

QUALCOMM

(i) 247.6 + 359.4

247.6	0101 0111 1010 .1001	(In XS-3 code)
+ 359.4	0110 1000 1100 .0111	
607.0	1011 1111 0110 1 0000	(Carry generated)
	+1 +1 ~~000~~	(propagate carry)
	1011 1 0000 0111 .0000	
	+1	
	1100 0000 0111 .0000	(0000 is invalid so add 0011 to 000 and 0111)
	−0011 +0011 +0011 +.0011	(Substract 0011 from 1100 to correct it)
	1001 0011 1010 .0011	(Corrected sum in XS-3)

Q. Substraction in XS-3 code

(i) 267 − 175

Soln:-

267	0101 1001 1010	
− 175	− 0001 1010 1000	
092	0000 1111 0010	(Correct 0010 and 0000 by adding 0011)
In XS-3 3(12)5	+0011 −0011 +0011	(Correct 1111 by substracting 0011)
	0011 1100 0101	Corrected difference in (XS-3)

(ii) 57.6 − 27.8

57.6	1000 1010 .1000	(In XS-3 code)
−27.8	− 0101 1010 .1011	
29.8	0010 1111 .1110	
{5(12).(11)}	+0011 −0011 −0011	
	0101 1100 .1011	Corrected difference in XS-3

Q. Subtraction in XS-3 code using 9's complement:

(i) 687 − 348

Solⁿ:- The subtrahend (348) in XS-3 code and its 9's complement are:

9's complement of 348 = 999 − 348 = 651

XS-3 code for 348 = 0110 0111 1011

1's complement of 348 in XS-3 = 1001 1000 0100

this is XS-3 code of 687

```
  687            687
- 348   =>    + 651   (9's complement of 348)
  339          (1)338
                ↳ +1   (End around carry)
                 339
```

```
   1001    1011    1010   (687 in XS-3)
+  1001    1000    0100   (1's complement of 348 in XS-3)
(1)0010 (1)0011    1110   (Carry generated)
 +1      +1                (Propagate carry)
   0011    0011    1110
                      +1  (End around carry)
   0011    0011    1111   To correct 1111 sub 0011
 +0011   +0011   -0011
   0110    0110    1100   (Correct difference)
```

(ii) 246 − 592

9's complement of 592 is 407

XS-3 code of 592 = 1000 1100 0101

1's complement of 592 = 0111 0011 1010

XS-3 code of 246 = 0101 0111 1001

```
  246            246
- 592   =>    + 407   (9's complement of 592)
- 346           653   (No carry)
```

So the result is negative and is in 9's complement form

The 9's Complement of 653 is 346.
So the result is −346.

```
  0101  0111  1001
+ 0111  0011  1010
  1100  1010  00011
            +1
  1100  1011  0011
-0011 -0011 +0011
  1001  1000  0110   (No carry)
```

So 1's complement of the sum is
0110 0111 1001 in XS-3 code (346_{10})
So Answer is −346

Q. Subtractions in XS-3 code using 10's Complement
(i) 597 − 239
10's Complement of 239 is 1000−239 = 761
XS-3 code of 239 = 0101 0110 1100
2's Complement of 239 = 1010 1001 0100
XS-3 code of 597 = 1000 1100 1010

```
  597          597
− 239   ⇒    + 761   (10's Complement of 239 in XS-3)
  358        (1)358   (ignor carry)
```

```
  1000  1100  1010   (597 in XS-3)
+ 1010  1001  0100   (2's Complement of 239 in XS-3)
(1)0010 (1)0101  1110   (Propagate carry)
+1      +1
(1) 0011  0101  1110   (Ignor carry)
+0011 +0011 −0011
  0000  0000  1011   (Corrected difference in XS-3 code)
```

There is carry so result is positive no.

(ii) 354 − 679

10's Complement of 672 is 328

XS-3 code of 672 is 1001 1010 0101

2's Complement of 679 is 0110 0101 1011

XS-3 code of 354 is 0110 1000 0111

$$\begin{array}{r} 354 \\ -\ 672 \\ \hline -318 \end{array} \Rightarrow \begin{array}{r} 354 \\ +\ 328 \\ \hline 682 \end{array}$$ (No carry)

So the result is negative and it is 10's Complement of 682 so, −318 is answer.

$$\begin{array}{rrrl} 0110 & 1000 & 0111 & \text{(XS-3 code of 354)} \\ +\ 0110 & 0101 & 1011 & \text{(2's Complement of 672 in XS-3)} \\ \hline 1100 & 1101 & 10010 & \\ & +1 & & \text{(Propagate carry)} \\ \hline 1100 & 1110 & 0010 & \\ -0011 & -0011 & +0011 & \text{(No carry = subtract 0011, carry = add 0011)} \\ \hline 1001 & 1011 & 0101 & \text{(No carry)} \end{array}$$

2's Complement of Sum 10011011 0101 is 0110 0100 1011 in XS-3 code (318_{10})

So the Answer is −318.

Error-Detecting Codes :—

when a binary data is transmitted and processed there is a possibility of distortion of its contents due to noise in channel.

The 1s may get changed to 0s and 0s to 1s.

In digital system error in transmitted data pose a serious problem.

So that whenever such an error occurs the concerned binary word can be corrected and retransmitted.

Parity —

The simplest technique for detecting error is that of adding an extra bit, known as the parity bit, to each word being transmitted.

There are two types of parity - odd & even parity

odd parity — the parity bit is set to a 0 or a 1 at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd number.

Even parity — the parity bit is set to a 0 or 1 at the transmitter such that the total no. of 1 bits in the word including the parity bit is an even number.

odd & Even parity in 8421 BCD code.

Decimal digit	8421 BCD	Odd parity	Even parity
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0101	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

When the digital data is received, a parity checking circuit generates an error signal if the total no. of 1s is even in odd-parity system or odd in an even parity system.

This parity check can always detect a single-bit error but cannot detect two or more errors within the same word.

The propability that two or more errors will occur simultaneously, although non zero, is substantially smaller.

Odd parity is used ~~to~~ more often than even parity because even parity does not detect the situation where all 0s are created by short ckt.

EDCodes - ~~are~~ ~~used~~ have the ability to detect any single bit error which transforms a valid code word into an invalid one.

A code is said to be an error-detecting code if its minimum distance is two or more.

The distance between two words is defined as the number of digits that must change in a word so that the other word results.

Ex:- The distance between 0011 and 1010 is 2
The distance between 0111 and 1000 is 4.

Ex:- In an even-parity scheme, which of the following words contain an error?

(i) 10101010
The no. of 1s in the word is even i.e (4). Therefore, there is no error

(ii) 10111001
The no. of 1s in the word is odd i.e 5.
Therefore, this word has an error.

QUALCOMM

Ex:- In an odd-parity scheme, which of the following words contains an error?

i) 10110111

The numer of 1s in the word is even i.e 6 so this word has error.

ii) 11101010

The no. of 1s in the word is odd i.e 5 Therefor, there is no error.

Check Sums —

→ Simple parity cannot detect two errors within the same word.

→ To overcome this problem we can use a sort of two-dimensional parity.

→ As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end.

At the end of transmission, the sum (called the Check Sum) up to that time is sent to the receiver.

The receiver can check its sum with the transmitted sum.

if the two sums are the same, then no errors were detected at the receiver end.

If there is an error, the receiver can ask for retransmission of the entire data.

This type of transmission used in tele processing system.

Block parity –

When several binary words are transmitted or stored in succession, the resulting collection of bits can be regarded as a block of data, having rows and columns.

Parity bits can then be assign to both rows and columns. This scheme makes it possible to correct any single bit error and to detect any two bit error in a word.

This technique is widely used for data stored on magnetic tapes.

Example:– Six 8bit succession can be formed into a 6X8 block for transmission.

Parity bits are added to both row & column so that the block is transmitted as 7X9 block as shown in Figure A.

At the receiving end, parity is checked both row and column wise, suppose errors are detect and corrected by complementing the error bit, shown in fig B 3rd row and 5th column means 5th bit in 3rd row is in error. It can bet corrected by complementing it.

Two error shown in figure C can only detected but not corrected, parity error observed in both column 2 and 4. It indicate that in one row there are two error.

Figure 2 TABLE A

ODD parity

	0	1	0	1	1	0	1	1	0
	1	0	0	1	0	1	0	1	1
	0	1	1	0	1	1	1	0	0
	1	1	0	1	0	0	1	1	0
	1	0	0	0	1	1	0	1	1
Parity	0	1	1	1	0	1	1	1	1
Row →	0	1	1	1	0	1	1	0	0

↑ Parity Column.

QUALCOMM

TABLE B

									P
	0	1	0	1	1	0	1	1	0
	1	0	0	1	0	1	0	1	1
	0	1	1	0	0	1	1	0	0
	1	1	0	1	0	0	1	1	0
	1	0	0	0	1	1	0	1	1
	0	1	1	1	0	1	1	1	1
P →	0	1	1	1	0	1	1	0	0

Error Bit

← 3rd Row

↑ 5th column

TABLE C

									P
	0	1	0	1	1	0	1	1	0
	1	0	0	1	0	1	0	1	1
	0	1	1	0	1	1	1	0	0
	1	0	0	0	0	0	1	1	0
	1	0	0	0	1	1	0	1	1
	0	1	1	1	0	1	1	1	1
P →	0	1	1	1	0	1	1	0	0

↑ ↑

Parity error in 2nd and 4th columns

Five-Bit Codes —

There are some 5-bit BCD code that have parity contained within each code for ease of error detection shown in Table.

(i) The 63210 is a weighted code (except for the decimal digit zero). It has the useful error detecting property that there are exactly two 1s in each code group. This code is used for storing data on magnetic tapes.

(ii) The 2-out of-5 code is a non-weighted code. It has also exactly two 1s in each code group. This code is used in the telephone and communication

inductries. At the receiving end, the receiver can check the no. of 1s in each character received.

(iii) The Shift Counter code, also called Johnson code, has the bit patterns produced by a 5-bit Johnson counter.

(iv) The 51111 code is similar to Johnson code but is weighted.

Five-bit BCD codes

Decimal	6 3 2 1 0	2-out-of-5 (0 1 2 3 4)	Shift counter (7 4 2 1 0)	5 1 1 1 1
0	0 0 1 1 0	0 0 0 1 1	0 0 0 0 0	0 0 0 0 0
1	0 0 0 1 1	0 0 1 0 1	0 0 0 0 1	0 0 0 0 1
2	0 0 1 0 1	0 0 1 1 0	0 0 0 1 1	0 0 0 1 1
3	0 1 0 0 1	0 1 0 0 1	0 0 1 1 1	0 0 1 1 1
4	0 1 0 1 0	0 1 0 1 0	0 1 1 1 1	0 1 1 1 1
5	0 1 1 0 0	0 1 1 0 0	1 1 1 1 1	1 0 0 0 0
6	1 0 0 0 1	1 0 0 0 1	1 1 1 1 0	1 1 0 0 0
7	1 0 0 1 0	1 0 0 1 0	1 1 1 0 0	1 1 1 0 0
8	1 0 1 0 0	1 0 1 0 0	1 1 0 0 0	1 1 1 1 0
9	1 1 0 0 0	1 1 0 0 0	1 0 0 0 0	1 1 1 1 1

The Biquinary code —

It is a weighted 7-bit BCD code. It is a parity data code each code group regarded as consisti of 2 bit subgroup and a 5-bit subgroup and each of these subgroups contains a single 1s. Thus it has the error checking feature, for each code group has exactly two 1s and each subgroup has exactly one 1.

The weights of the bit positions are 5043210.

The biquinary code is used in Abacus.

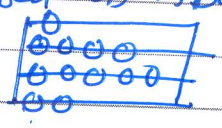


TABLE The Biquinary Code.

Decimal	Biquinary code 5	0	4	3	2	1	0
0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0
7	1	0	0	0	1	0	0
8	1	0	0	1	0	0	0
9	1	0	1	0	0	0	0

Ring-Counter code -

A 10-bit ring counter code produce a sequence of 10-bit group having the property that each group has a single 1. It is a weighted code because each bit weights equals to 1 of the 10 decimal digits.

Although this code is inefficient (for 1024 encode in pure binary), it has excellent error detecting property to implement Ring counter

Ring counter code

Decimal	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0

Error - Correcting Codes -

- Error correcting codes are used to correct the errors present in the received data (bit stream).
- Parity code is also used for error correction.
- To correct a single bit error the minimum distance of the code must be three.
- The error correction code must be locate the error bit (k bit position) and by complementing it, the message can be corrected.

Hamming code is the example of error correcting code.

- Hamming code is useful for both detection and correction of error present in the received data.
- This code uses multiple parity bits ~~and~~ $P_1, P_2 \ldots P_k$ located at positions 2^{k-1} such as $2^0, 2^1, 2^2, 2^3 \ldots$ so on.
- The minimum value of 'k' for which the following relation is correct (valid) is nothing but the required no. of parity bits. $2^k \geq n + k + 1$

  where, 'n' is the no. of bits in the binary code 'k' is the no. of parity bits.
- Therefore the no. of bits in the Hamming code is equal to $(n+k)$ bit code word.
- To correct the error, k parity checks are performed on selected digits of each code word and the position of the error bit is located by forming an "error word" and the error bit is then complemented.
- The k-bit error word is generated by putting 0 or 1 in 2^{k-1} position depending upon parity type (Even or odd parity). For even $P=0$, for odd $P=1$

7-bit Hamming code - (n,k) (7,3)

Hamming code structure -

All bit positions that are power of 2 are marked as parity bits ($2^0, 2^1, 2^2, 2^3 \ldots$) other bits are for data bits.

The word format of 7-bit HC would be as shown below:-

D7	D6	D5	P4	D3	P2	P1

Determine the value of parity bit :-

Rule 1- The value of parity bit is determine by the sequence of bits that is alternatively checks and skips. Sender $\xrightarrow{1101}$ Receiver

D7	D6	D5	P4	D3	P2	P1
1	1	0	?	1	?	?

For P_1 :- Check 1 bit, skip 1 bit, check 1 bit, skip 1 bit and so on
(1, 3, 5, 7, 9 - - - -)

For P_2 : Check 2 bit, skip 2 bit, check 2 bit, skip 2 bit
(2, 3, 6, 7, 10, 11 - - -)

For P_4 : Check 4 bit, skip 4 bit, check 4 bit, skip 4 bit and so on
(4, 5, 6, 7, 12, 13, 14, 15 - - - -)

For Even parity bit -

P_1 D_3 D_5 D_7	P_2 D_3 D_6 D_7	P_4 D_5 D_6 D_7
P_1 1 0 1	P_2 1 1 1	P_4 0 1 1
$P_1 = 0$	$P_2 = 1$	$P_4 = 0$

∴ So the 7-bit hamming code is

D7	D6	D5	P4	D3	P2	P1
1	1	0	0	1	1	0

Error detection :-

Consider a 7 bit Hamming code as
$D_7\ D_6\ D_5\ P_4\ D_3\ P_2\ P_1$
At receiver end,
bit (1,3,5,7), (2,3,6,7) and (4,5,6,7) are checked for even parity.
if $P_1 = 0$, $P_2 = 0$, $P_4 = 0$ so there is no error.
if any of P_1, P_2, P_4 is equal to 1 then there is error.

Error correction :-

An error is located by forming a 3 bit no. out of 3 parity checks. $\boxed{P_4 | P_2 | P_1}$

Finding P_1 :-
we check parity of $(P_1, P_3\ D_5, D_7)$
if it is odd error exist $P_1 = 1$
even : no error $P_1 = 0$
Similarly for P_2 & P_4

After we have found the error word, we find its decimal value.
Then we complement the incorrect bit at the decimal value position.

Que :- Assume that the data has been encoded in a 7-bit even-parity Hamming code and the number [1011011] is received. Find out the bit in error. What will be the corrected code?

Soln :- The 7-bit Even parity Hamming code received at receiver end is -

D_7	P_6	D_5	P_4	D_3	P_2	P_1
1	0	1	1	0	1	1

Error detection -
Analyzing the bits 1, 3, 5, 7 (1 0 1 1) of which P_1 is a parity bit, an error is detected.

QUALCOMM

So $P_1 = 1$ because for odd parity

Analysing the bits {2, 3, 6, 7} (1 0 0 1) of which P_2 is a parity bit, no. of 1s is even so there is no error. Hence $P_2 = 0$

Analysing the bit stream 4, 5, 6, 7 (1 1 0 1) of which P_4 is a parity bit, no. of 1s is odd so an error is detected.

Hence $P_4 = 1$

So Error word is

P_4	P_2	P_1
1	0	1

The decimal equivalent of (101) is 5. so the 5th bit is an error bit.

Error correction –

By complementing 5th from 1 to 0 it get corrected. So the corrected received data word is (1 0 0 1 0 1 1).

Alphanumeric Codes :-

Alphanumeric codes are codes used to encode the characters of alphabet in addition to the decimal digits.

They are used for transmitting data between computers and its I/O devices such as printers, keyboards and video display terminals.

The no. of bits used in most alphanumeric codes is much more than those required to encode 10 decimal digits and 26 alphabetic characters, these codes include bit patterns for a wide range of other symbols and functions as well.

i) ASCII code - American Standard Code for Information Interchange pronounced as 'ASKEE' is a widely used alphanumeric code. This is basically a 7-bit code. So there is $2^7 = 128$ different bit patterns.

The ASCII can be used to encode both the lowercase and uppercase character of the alphabet (52 symbols) and some special symbols (&, #, $, &, %) as well, in addition to the 10 decimal digits.

Because characters are assigned in ascending binary no.s, ASCII is very easy for computer to alphabetize and sort.

Abbreviations -

ACK	Acknowledge	DEL	Delete idle
BEL	Bell	DLE	Data link escape
BS	Backspace	EM	End of medium
CAN	Cancel	ENQ	Enquiry
CR	carriage return	EOT	End of transmission
DC1	Direct control 1	ESC	Escape
DC2	Direct control 2	ETB	End of transmission Block
DC3	Direct control 3	EXT	End of Text
DC4	Direct control 4	FF	Form Feed
		FS	Form separator

GS	Group Seprator	SO	Shift out
HT	Horizontal tab	SOH	Start of heading
LF	Line Feed	STX	start text
NAK	Negaline Acknowledge	SUB	Substitute
NUL	Null	SYN	Synchronous idle
RS	Record separator	US	unit separator
SI	Shift in	VT	vertical tab

The ASCII code

LSB \ MSB	000	001	010	011	100	101	110	111
0000	NUL	DEL	SPACE	0	@	P		p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	END	~~NAK~~	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	—	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DLE

The EBCDIC Code –

Extended Binary coded Decimal Interchange code pronounced as 'eb-si-dik' is an 8 bit alphanumeric code. ($2^8 = 256$) bit patterns can be formed.

It can be used to encode all symbols and control characters found in ASCII and other symbols.

In facts, many of the bit patterns in the EBCDIC code are unassigned.

This code uses BCD as the basis of binary assignment. Used in computer.

The EBCDIC code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&							[	]	\	0
1	SOH	DC1	SOS				/		a	j	~		A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	EOB	YC					f	o	w		F	O	W	6
7	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z		I	R	Z	9
A	SMM	CC	SM		¢	!	|	:								
B	VT				.	$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	–	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	|	¬	?	"								